New algorithms for relaxed multiplication

Joris van der Hoeven

Dépt. de Mathématiques (Bât. 425)
CNRS, Université Paris-Sud
91405 Orsay Cedex
France
Email: `joris@texmacs.org`

Abstract

In previous work, we have introduced the technique of relaxed power series computations. With this technique, it is possible to solve implicit equations almost as quickly as doing the operations which occur in the implicit equation. Here "almost as quickly" means that we need to pay a logarithmic overhead. In this paper, we will show how to reduce this logarithmic factor in the case when the constant ring has sufficiently many 2^p-th roots of unity.

Key words: power series, multiplication, algorithm, FFT, computer algebra

1 Introduction

Let $\mathsf{C} \ni \{\frac{1}{2}\}$ be an effective ring and consider two power series $f = f_0 + f_1 z + \cdots$ and $g = g_0 + g_1 z + \cdots$ in $\mathsf{C}[[z]]$. In this paper we will be concerned with the efficient computation of the first n coefficients of the product $h = fg = h_0 + h_1 z + \cdots$.

If the first n coefficients of f and g are known beforehand, then we may use any fast multiplication for polynomials in order to achieve this goal, such as divide and conquer multiplication (Karatsuba and Ofman, 1963; Knuth,

[⋆] The paper was originally written using GNU $\TeX_{\text{MACS}}$ (see `www.texmacs.org`). Unfortunately, Elsevier insists on the use of $\LaTeX$ with its own style files. Any insufficiencies in the typesetting quality should therefore be imputed to Elsevier.

1997), which has a time complexity $K(n) = O(n^{\log 3/\log 2})$, or F.F.T. multiplication (Cooley and Tukey, 1965; Schönhage and Strassen, 1971; Cantor and Kaltofen, 1991; van der Hoeven, 2002), which has a time complexity $M(n) = O(n \log n \log \log n)$.

For certain computations, and most importantly the resolution of implicit equations, it is interesting to use so called "relaxed algorithms" which output the first i coefficients of h as soon as the first i coefficients of f and g are known for each $i \leqslant n$. This allows for instance the computation of the exponential $g = \exp f$ of a series f with $f_0 = 0$ using the formula

$$g = \int f' g. \tag{1}$$

More precisely, this formula shows that the computation of $\exp f$ reduces to one differentiation, one relaxed product and one relaxed integration. Differentiation and relaxed integration being linear in time, it follows that n terms of $\exp f$ can be computed in time $R(n) + O(n)$, where $R(n)$ denotes the time complexity of relaxed multiplication. In (van der Hoeven, 1997, 2002), we proved the following theorem:

Theorem 1 *There exists a relaxed multiplication algorithm of time complexity*

$$R(n) = O(M(n) \log n)$$

and space complexity $O(n)$.

In this paper, we will improve the time complexity bound in this theorem in the case when C admits 2^p-th roots of unity for any $p \in \mathbb{N}$. In section 2, we first reduce this problem to the case of "semi-relaxed multiplication", when one of the arguments is fixed and the other one relaxed. More precisely, let f and g be power series, such that g is known up to order n. Then a semi-relaxed multiplication algorithm computes the product $h = fg$ up to order n and outputs $(fg)_i$ as soon as $f_0, \ldots, f_i$ are known, for all $i < n$. In section 3, we show that the $\log n$ overhead in theorem 1 can be reduced to $O((\log n)^{\log 3/\log 2})$. In section 4, the technique of section 3 is further improved so as to yield an $O(\mathrm{e}^{2\sqrt{\log 2 \log \log n}})$ overhead.

In the sequel, we will use the following notations from (van der Hoeven, 2002): we denote by $\mathsf{C}[[z]]_n \subseteq \mathsf{C}[z] \subseteq \mathsf{C}[[z]]$ the set of truncated power series of order n, like $f = f_0 + \cdots + f_{n-1} z^{n-1}$. Given $f \in \mathsf{C}[[z]]_n$ and $0 \leqslant i < j \leqslant n$, we will denote $f_{i \ldots j} = f_i + \cdots + f_{j-1} z^{j-i-1} \in \mathsf{C}[[z]]_{j-i}$.

Remark 1 An preprint of the present paper was published a few years ago (van der Hoeven, 2003a). The current version includes a new section 5 with implementation details, benchmarks and a few notes on how to apply similar ideas in the Karatsuba and Toom-Cook models. Another algorithm for semi-relaxed multiplication, based on the middle product (Hanrot et al., 2004), was also published before (van der Hoeven, 2003b).

Remark 2 The exotic form $O(n \log n \mathrm{e}^{2\sqrt{\log 2 \log \log n}})$ of the new complexity for relaxed multiplication might surprise the reader. It should be noticed that the time complexity of Toom-Cook's algorithm for polynomial multiplication (Toom, 1963; Cook, 1966) has a similar complexity $O(n \log n 2^{\sqrt{2 \log n}})$ (Knuth, 1997, Section 4.3, p. 286 and exercise 5, p. 300). Indeed, whereas our algorithm from section 3 has a Karatsuba-like flavour, the algorithm from section 4 uses a generalized subdivision which is similar to the one used by Toom and Cook.

An interesting question is whether even better time complexities can be obtained (in analogy with FFT-multiplication). However, we have not managed so far to reduce the cost of relaxed multiplication to $O(M(n))$ or another sharper complexity such as $O(M(n) \log \log \log n)$. Nevertheless, it should be noticed that the function $\mathrm{e}^{2\sqrt{\log 2 \log \log n}}$ grows very slowly; in practice, it very much behaves like as a constant (see section 5).

Remark 3 The reader may wonder whether further improvements in the complexity of relaxed multiplication are really useful, since the algorithms from (van der Hoeven, 1997, 2002) are already optimal up to a factor $O(\log n)$. In fact, we expect fast algorithms for formal power series to be one of the building bricks for effective analysis (van der Hoeven, 2006b). Therefore, even small improvements in the complexity of relaxed multiplication should lead to global speed-ups for this kind of software.

2 Full and semi-relaxed multiplication

In (van der Hoeven, 1997, 2002), we have stated several fast algorithms for relaxed multiplication. Let us briefly recall some of the main concepts and ideas. For details, we refer to (van der Hoeven, 2002). Throughout this section, f and g are two power series in $\mathsf{C}[[z]]$.

Definition 1 *We call*

$$P = f_{0 \ldots n} g_{0 \ldots n} \tag{2}$$

the full product of f and g at order n.

Definition 2 *We call*

$$P = \sum_{i+j<n} (f_i g_j) z^{i+j} \tag{3}$$

the truncated product of f and g at order n.

Definition 3 *A full (or truncated) zealous multiplication algorithm of f and g at order n takes $f_0, \ldots, f_{n-1}$ and $g_0, \ldots, g_{n-1}$ on input and computes P as in (2) (resp. (3)).*

Definition 4 *A full (or truncated) relaxed multiplication algorithm of f and g at order n successively takes the pairs $(f_0, g_0), \ldots, (f_{n-1}, g_{n-1})$ on input and successively computes $P_0, \ldots, P_{2n-2}$ (resp. $P_0, \ldots, P_{n-1}$). Here it is understood that P_i is output as soon as $(f_0, g_0), \ldots, (f_i, g_i)$ are known.*

Definition 5 *A full (or truncated) semi-relaxed multiplication algorithm of f and g takes $g_0, \ldots, g_{n-1}$ and the successive values $f_0, \ldots, f_{n-1}$ on input and successively computes $P_0, \ldots, P_{2n-2}$ (resp. $P_0, \ldots, P_{n-1}$). Here it is understood that P_i is output as soon as $f_0, \ldots, f_i$ are known.*

We will denote by $M(n)$, $R(n)$ and $Q(n)$ the time complexities of full zealous, relaxed and semi-relaxed multiplication at order n, where it is understood that the ring operations in C can be performed in time $O(1)$. We notice that full zealous multiplication is equivalent to polynomial multiplication. Hence, classical fast multiplication algorithms can be applied in this case (Karatsuba and Ofman, 1963; Toom, 1963; Cook, 1966; Cooley and Tukey, 1965; Schönhage and Strassen, 1971; Cantor and Kaltofen, 1991; van der Hoeven, 2002).

The main idea behind efficient algorithms for relaxed multiplication is to anticipate on future computations. More precisely, the computation of a full product (2) can be represented by an $n \times n$ square with entries $f_i g_j$, $0 \leqslant i, j < n$. As soon as $f_0, \ldots, f_i$ and $g_0, \ldots, g_i$ are known, it becomes possible to compute the contributions of the products $f_j g_k$ with $0 \leqslant j, k \leqslant i$ to P, even though the contributions of $f_j g_k$ with $j + k > i$ are not yet needed. The next idea is to subdivide the $n \times n$ square into smaller squares, in such a way that the contribution of each small square to P can be computed using a zealous algorithm. Now the contribution of such a small square is of the form $f_{i_1 \ldots i_2} g_{j_1 \ldots j_2} z^{i_1 + j_1}$. Therefore, the requirement $i_1 + j_1 \leqslant \max(i_2, j_2)$ suffices to ensure that the resulting algorithm will be relaxed. In the left hand image of figure 1, we have shown the subdivision from the main algorithm of (van der Hoeven, 1997, 2002), which has time complexity $R(n) = O(M(n) \log n)$.

There is an alternative interpretation of the left hand image in figure 1: when interpreting the big square as a $2n \times 2n$ multiplication

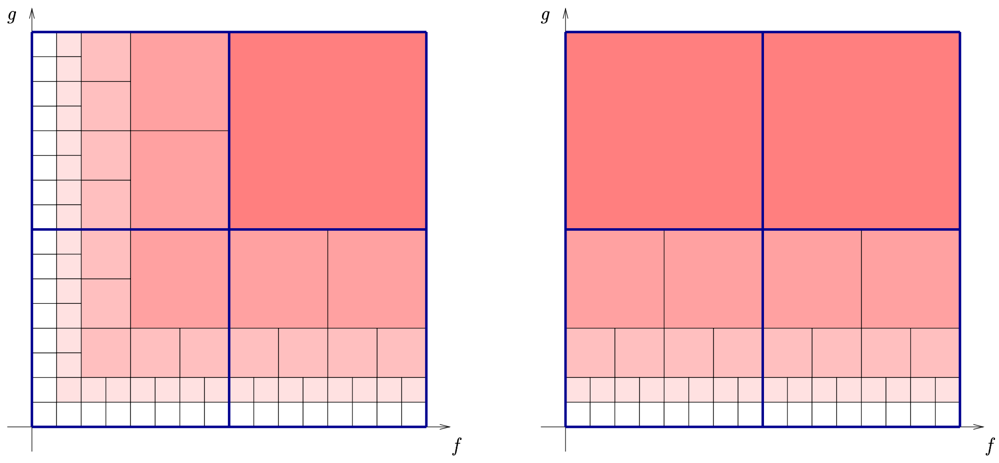

Fig. 1. Illustration of the facts that (1) a full relaxed $2n \times 2n$ multiplication reduces to one full relaxed $n \times n$ multiplication, two semi-relaxed $n \times n$ multiplication and one zealous $n \times n$ multiplication (2) a semi-relaxed $2n \times 2n$ multiplication reduces to two semi-relaxed $n \times n$ multiplications and two zealous $n \times n$ multiplications.

$$P = f_{0\ldots 2n} g_{0\ldots 2n},$$

we may regard it as the sum

$$P = P_{0,0} + P_{0,1} z^n + P_{1,0} z^n + P_{1,1} z^{2n}$$

of four $n \times n$ multiplications

$$\begin{aligned} P_{0,0} &= f_{0\ldots n} g_{0\ldots n} \\ P_{0,1} &= f_{0\ldots n} g_{n\ldots 2n} \\ P_{1,0} &= f_{n\ldots 2n} g_{0\ldots n} \\ P_{1,1} &= f_{n\ldots 2n} g_{n\ldots 2n}. \end{aligned}$$

Now $P_{0,0}$ is a relaxed multiplication at order n, but $P_{0,1}$ is even semi-relaxed, since $g_0, \ldots, g_{n-1}$ are already known by the time that we need $(P_{0,1})_0$. Similarly, $P_{1,0}$ corresponds to a semi-relaxed product and $P_{1,1}$ to a zealous product. This shows that

$$R(2n) \leqslant R(n) + 2Q(n) + M(n).$$

Similarly, we have

$$Q(2n) \leqslant 2Q(n) + 2M(n),$$

as illustrated in the right-hand image of figure 1. Under suitable regularity hypotheses for $M(n)$ and $Q(n)$, the above relations imply:

Theorem 2

a) If $\frac{M(n)}{n}$ is increasing, then $Q(n) = O(M(n) \log n)$.
b) If $\frac{Q(n)}{n}$ is increasing, then $R(n) = O(Q(n))$.

A consequence of part (b) of the theorem is that it suffices to design fast algorithms for semi-relaxed multiplication in order to obtain fast algorithms for relaxed multiplication. This fact may be reinterpreted by observing that the fast relaxed multiplication algorithm actually applies Newton's method in a hidden way. Indeed, since Brent and Kung (Brent and Kung, 1978), it is well known that Newton's method can also be used in the context of formal power series in order to solve differential or functional equations. One step of Newton's method at order n involves the recursive application of the method at order $\lceil n/2 \rceil$ and the resolution of a linear equation at order $\lfloor n/2 \rfloor$. The resolution of the linear equation corresponds to the computation of the two semi-relaxed products.

3 A new algorithm for fast relaxed multiplication

Assume from now on that C admits an n-th root of unity ω_n for every power of two $n \in 2^{\mathbb{N}}$. Given an element $f \in \mathsf{C}[[z]]_n$, let $\mathrm{FFT}_n(f) \in \mathsf{C}^n$ denote its Fourier transform

$$\mathrm{FFT}_n(f) = (f(1), f(\omega_n), \ldots, f(\omega_n^{n-1}))$$

and let $\mathrm{FFT}_n^{-1} : \mathsf{C}^n \to \mathsf{TPS}(n)$ be the inverse mapping of FFT_n. It is well known that both FFT_n and FFT_n^{-1} can be computed in time $O(n \log n)$. Furthermore, if $f, g \in \mathsf{C}[[z]]_n$ are such that $fg \in \mathsf{C}[[z]]_n$, then

$$fg = \mathrm{FFT}_n^{-1}(\mathrm{FFT}_n(f)\, \mathrm{FFT}_n(g)),$$

where the product in C^n is scalar multiplication $(a_0, \ldots, a_{n-1})(b_0, \ldots, b_{n-1}) = (a_0 b_0, \ldots, a_{n-1} b_{n-1})$.

Now consider a decomposition $n = n_1 n_2$ with $n_1 = 2^{p_1}$ and $n_2 = 2^{p_2}$. Then a truncated power series $f \in \mathsf{C}[z]_n$ can be rewritten as a series

$$f_{0 \ldots n_1} + f_{n_1 \ldots 2n_1} y + \cdots + f_{(n_2-1)n_1 \ldots n_2 n_1} y^{n_2-1}$$

in $\mathsf{C}[z]_{n_1}[y]_{n_2}$, where $y = z^{n_1}$. This series may again be reinterpreted as a series $\mathrm{N}(f) \in \mathsf{C}[z]_{2n_1}[y]_{n_2}$, and we have

$$fg = \mathrm{N}^{-1}(\mathrm{N}(f)\mathrm{N}(g)),$$

where $\mathrm{N}^{-1} : \mathsf{C}[z]_{2n_1}[y] \to \mathsf{C}[z]$ is the mapping which substitutes z^{n_1} for y. Also, the FFT-transform $\mathrm{FFT}_{2n_1} : \mathsf{C}[z]_{2n_1} \to \mathsf{C}^{2n_1}$ may be extended to a mapping

$$\begin{aligned} \mathsf{C}[z]_{2n_1}[y]_l &\longrightarrow \mathsf{C}^{2n_1}[y]_l \\ c_0 + \cdots + c_{l-1}y^{l-1} &\longmapsto \mathrm{FFT}_d(c_0) + \cdots + \mathrm{FFT}_d(c_{l-1})y^{l-1} \end{aligned}$$

for each l, and similarly for its inverse $\mathrm{FFT}_{2n_1}^{-1}$. Now the formula

$$fg = \mathrm{N}^{-1}(\mathrm{FFT}_{2n_1}^{-1}(\mathrm{FFT}_{2n_1}(\mathrm{N}(f))\,\mathrm{FFT}_{2n_1}(\mathrm{N}(g))))$$

yields a way to compute fg by reusing the Fourier transforms of the "bunches of coefficients" $f_{kn_1\ldots(k+1)n_1}$ and $g_{ln_1\ldots(l+1)n_1}$ many times.

In the context of a semi-relaxed multiplication fg with fixed argument g, the above scheme almost reduces the computation of an $n \times n$ product with coefficients in C to the computation of an $n_2 \times n_2$ product with coefficients in C^{2n_1}. The only problem which remains is that $\mathrm{FFT}_{2n_1}(f_{kn_1\ldots(k+1)n_1})$ can only be computed when $f_{kn_1}, \ldots, f_{(k+1)n_1-1}$ are all known. Consequently, the products $f_{kn_1\ldots(k+1)n_1}g_{0\ldots n_1}$ should be computed apart, using a traditional semi-relaxed multiplication. In other words, we have reduced the computation of a semi-relaxed $n\times n$ product with coefficients in C to the computation of n_2 semi-relaxed $n_1 \times n_1$ products with coefficients in C, one semi-relaxed $n_2 \times (n_2 - 1)$ product with coefficients in C^{2n_1} and $4n_2 - 3$ FFT-transforms of length $2n_1$. This has been illustrated in figure 2.

In order to obtain an efficient algorithm, we may choose $p_1 = \lceil p/2 \rceil$ and $p_2 = \lfloor p/2 \rfloor$:

Theorem 3 *Assume that* C *admits an* n*-th root of unity for each* $n \in 2^{\mathbb{N}}$*. Then there exists a relaxed multiplication algorithm of time complexity*

$$O(n(\log n)^{\log 3/\log 2})$$

and space complexity

$$O(n \log n).$$

Proof In view of section 2, it suffices to consider the case of a semi-relaxed product. Let $T(n)$ denote the time complexity of the above method. Then we observe that

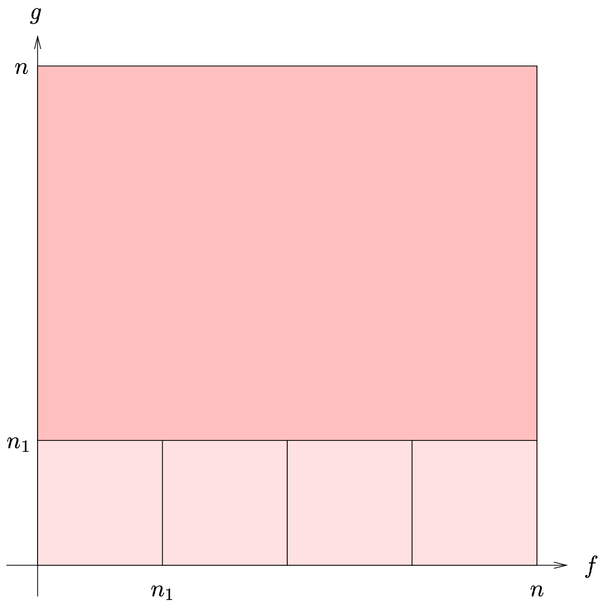

Fig. 2. New decomposition of a semi-relaxed $n \times n$ multiplication into n/n_1 semi-relaxed $n_1 \times n_1$ multiplications (the light regions) and one semi-relaxed $n_2 \times (n_2 - 1)$ multiplication (the dark region) with FFT-ed coefficients in C^{2n_1}.

$$\begin{aligned} T(n) &\leqslant n_2 T(n_1) + 2n_1 T(n_2) + O(n_2 n_1 \log n_1) \\ &\leqslant n_2 T(n_1) + 2n_1 T(n_2) + O(n \log n). \end{aligned}$$

Taking $p_1 = \lfloor p/2 \rfloor$, $p_2 = \lceil p/2 \rceil$ and $U(p) = T(2^p)/2^p$, we obtain

$$U(p) \leqslant U(\lceil p/2 \rceil) + 2U(\lfloor p/2 \rfloor) + O(p),$$

from which we deduce that $U(p) = O(p^{\log 3/\log 2})$ and

$$T(n) = O(n(\log n)^{\log 3/\log 2}).$$

Similarly, the space complexity $S(n)$ satisfies the bound

$$S(n) \leqslant S(n_1) + 2n_1 S(n_2) + O(n) \leqslant (2n_1 + 1)S(n_2) + O(n).$$

Setting $R(p) = S(2^p)/2^p$, it follows that

$$R(p) \leqslant (2 + \frac{1}{2^{\lfloor p/2 \rfloor}})R(\lceil p/2 \rceil) + O(1)$$

Consequently, $R(p) = O(p)$ and $S(n) = O(np) = O(n \log n)$. □

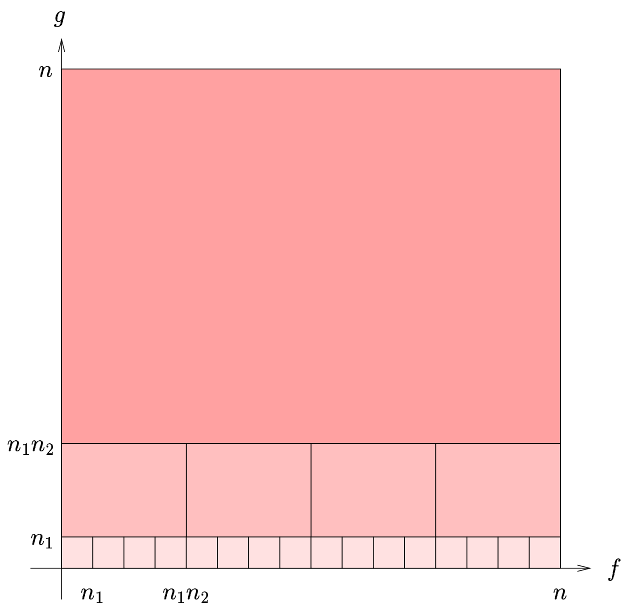

Fig. 3. Generalized decomposition of a semi-relaxed $n \times n$ multiplication into $l = 3$ layers.

4 Further improvements of the algorithm

More generally, if $n = n_1 \cdots n_l$ with $n_1 = 2^{p_1}, \ldots, n_l = 2^{p_l}$, then we may reduce the computation of a semi-relaxed $n \times n$ product with coefficients in C into the computation of

- $\frac{n}{n_1}$ semi-relaxed $n_1 \times n_1$ products over C of the form $f_{kn_1 \ldots (k+1)n_1} g_{0 \ldots n_1}$;
- $2(\frac{n}{n_1} + n_2 - 1) - 1$ FFT-transforms of length $2n_1$;
- $\frac{n}{n_1 n_2}$ semi-relaxed $n_2 \times (n_2 - 1)$ products over C^{2n_1};
- $2(\frac{n}{n_1 n_2} + n_3 - 1) - 1$ FFT-transforms of length $2n_1n_2$;
- $\frac{n}{n_1 n_2 n_3}$ semi-relaxed $n_3 \times (n_3 - 1)$ products over $\mathsf{C}^{2n_1 n_2}$;
- $\vdots$
- $4n_l - 3$ FFT-transforms of length $2\frac{n}{n_l}$;
- one semi-relaxed $n_l \times (n_l - 1)$ product over $\mathsf{C}^{2n_1 \cdots n_{l-1}}$.

This computation is illustrated in 3. From the complexity point of view, it leads to the following theorem:

Theorem 4 *Assume that* C *admits an* n*-th root of unity for each* $n \in 2^{\mathbb{N}}$*. Then there exists a relaxed multiplication algorithm of time complexity*

$$O(n \log n e^{2\sqrt{\log 2 \log \log n}})$$

and space complexity

$$O(n\mathrm{e}^{\sqrt{\log 2 \log\log n}}).$$

Proof In view of theorem $2(b)$, it suffices to consider the case of a semi-relaxed product. Denoting by $T(n)$ the time complexity of the above method, we have

$$T(n) \leqslant \frac{n}{n_1}T(n_1) + \frac{2n}{n_2}T(n_2) + \cdots + \frac{2n}{n_l}T(n_l) + O(ln\log n). \tag{4}$$

Let

$$U(p) = \frac{T(2^p)}{p2^p}.$$

Taking $n_1 = \cdots = n_l = 2^p$ in (4), it follows for any l that

$$U(lp) \leqslant 2U(p) + O(l). \tag{5}$$

Applying this relation k times, we obtain

$$U(l^k) \leqslant 2^k U(1) + O(2^k l) = O(2^k l). \tag{6}$$

For a fixed p such that $k = \log p/\log l$ is an integer, we obtain

$$U(p) = O(2^{\log p/\log l}l). \tag{7}$$

The minimum of $2^{\log p/\log l}l$ is reached when its derivative w.r.t. l cancels. This happens for

$$l_p = \mathrm{e}^{\sqrt{\log 2 \log p}}$$

Plugging this value into (7), we obtain

$$U(p) = O(\mathrm{e}^{2\sqrt{\log 2 \log p}}).$$

Substitution of $p = \log n/\log 2$ finally gives the desired estimate

$$T(n) = O(n\log n\mathrm{e}^{2\sqrt{\log 2 \log\log n}}). \tag{8}$$

In order to be painstakingly correct, we notice that we really proved (7) for p of the form $p = l^{\lceil \log p/\log l\rceil}$ and (8) for n of the form $n = 2^p$. Of course, we may always replace p and n by larger values which do have this form. Since these replacements only introduce additional constant factors in the complexity bounds, the bound (8) holds for general n.

As to the space complexity $S(n)$, we have

$$S(n) \leqslant S(n_1) + 2n_1 S(n_2) + \cdots + 2n_1 \cdots n_{l-1} S(n_l) + O(n).$$

Let

$$R(p) = \frac{S(2^p)}{2^p}.$$

Taking $n_1 = \cdots = n_l = 2^p$, it follows for any l that

$$R(lp) \leqslant (2 + C/2^p) R(p) + O(1),$$

for some fixed constant C. Applying this bound k times, we obtain

$$R(l^k) \leqslant \left(\prod_{i=1}^{k} 2 + \frac{C}{2^{il}} \right) (R(1) + O(1)).$$

For $l \to \infty$, this bound simplifies to

$$R(l^k) = O(2^k).$$

Taking $k = \log p / \log l$ and $l = \mathrm{e}^{\sqrt{\log 2 \log p}}$ as above, it follows that

$$R(p) = O(2^{\sqrt{\log p / \log 2}}) = O(\mathrm{e}^{\sqrt{\log 2 \log p}}).$$

Substitution of $p = \log n / \log 2$ finally gives us the desired estimate

$$S(n) = O(n \mathrm{e}^{\sqrt{\log 2 \log \log n}})$$

for the space complexity. For similar reasons as above, the bound holds for general n. □

5 Implementation details and benchmarks

We implemented the algorithm from section 3 in the C++ library Mmxlib (van der Hoeven et al., 2002). Instead of taking $n_1 \approx n_2$, we took n_2 small (with $n_2 \in \{4, 8, 16, 32\}$ in the FFT range up to $n = 2^{24}$), and used a naive multiplication algorithm on the FFT-ed blocks. The reason behind this change is that n_1 needs to be reasonably large in order to profit from the better asymptotic complexity of relaxed multiplication. In practice, the optimal choice of (n_1, n_2) is obtained by taking n_2 quite small.

Moreover, our implementation uses a truncated version of relaxed multiplication (van der Hoeven, 2002, Section 4.4.2). In particular, the use of naive multiplication on the FFT-ed blocks allows us to gain a factor 2 at the top-level. For small values of $n = 2^p$, we also replaced FFT transforms by "Karatsuba transforms": given a polynomial $f = f_0 + \cdots + f_{2^p-1} Z^{2^p-1}$, we may form a polynomial $F(Z_1, \ldots, Z_p)$ in p variables with coefficients $F_{i_0,\ldots,i_{p-1}} = f_{i_0+\cdots+i_{p-1}2^{p-1}}$ for $i_0, \ldots, i_{p-1} \in \{0, 1\}$. Then the Karatsuba transform of f is the vector $(F(z_0, \ldots, z_{p-1}))_{z_i \in \{0,1,\Omega\}}$ of size 3^p, where $(a + bZ)(\Omega) = b$.

We have both tested (truncated) relaxed and semi-relaxed multiplication for different types of coefficients on an Intel Xeon processor at 3.2 GHz with 1 Gb of memory. The results of our benchmarks can be found in tables 1 and 2 below. Our benchmarks start at the order n where FFT multiplication becomes useful. Notice that working with orders in $2^{\mathbb{N}}$ does not give us any significant advantage, because the top-level product on FFT-ed blocks is naive. In table 1, the choice of n_2 as a function of n has been optimized for complex double coefficients. No particular optimization effort was made for the coefficient types in table 2, and it might be possible to gain about 10% on our timings.

Remark 4 It is instructive to compare the efficiencies of relaxed evaluation and Newton's method. For instance, the exponentiation algorithm from (Brent and Kung, 1978) has a time complexity $\sim 4M(n)$. Although this is better from an asymptotic point of view, the ratio $Q(n)/M(n)$ rarely reaches 4 in our tables. Consequently, relaxed algorithms are often better. We already observed this phenomenon during our first implementation of exponentiation using both a relaxed algorithm and Newton's method (van der Hoeven, 2002, Tables 4 and 5). Nowadays, computers are faster and there have been some recent advances concerning Newton's method (Bostan et al., 2006; van der Hoeven, 2006a); see also (Sedoglavic, 2001, Section 5.2.1). Therefore, it would be interesting to carefully implement both methods and pursue the comparisons.

Remark 5 Although the emphasis of this paper is on asymptotic complexity, the idea behind the new algorithms also applies in the Karatsuba and Toom-Cook models. In the latter case, we take n_1 small (typically $n_1 \in \{2, 3, 4\}$) and use evaluation (interpolation) for polynomials of degree $n_1 - 1$ $(2n_1 - 2)$ at

n	$Q(n)$	$\frac{Q(n)}{M(n)}$	$R(n)$	$\frac{R(n)}{M(n)}$
2^8	0.001	1.844	0.001	1.923
2^9	0.003	2.266	0.003	2.633
2^{10}	0.007	2.426	0.008	2.879
2^{11}	0.014	2.377	0.017	2.878
2^{12}	0.031	2.537	0.037	3.037
2^{13}	0.068	2.659	0.088	3.385
2^{14}	0.158	2.844	0.190	3.420
2^{15}	0.341	2.893	0.437	3.701
2^{16}	0.767	3.038	1.018	4.032
2^{17}	1.703	3.151	2.195	4.061
2^{18}	3.618	2.968	4.618	3.770
2^{19}	8.097	3.001	10.319	3.820
2^{20}	17.307	2.921	22.149	3.723
2^{21}	37.804	2.916	49.347	3.856
2^{22}	80.298	2.881	104.159	3.746

Table 1
Timings in seconds for the computation of n terms of the exponential of a given series using complex double coefficients. We both computed the exponential using a semi-relaxed and a relaxed product, corresponding to $Q(n)$ and $R(n)$. We also considered the ratios with the timings $M(n)$ for a full FFT-product of two polynomials of degree $< n$.

$2n_1 - 1$ points. From an asymptotic point of view, this yields $R(n) \sim M(n)$ for relaxed multiplication. Moreover, the approach naturally combines with the generalization of pair/odd decompositions (Hanrot and Zimmermann, 2002), which also yields an optimal bound for truncated multiplications. In fact, we notice that truncated pair/odd Karatsuba multiplication is "essentially relaxed" (van der Hoeven, 2002, Section 4.2).

On the negative side, these theoretically fast algorithms have bad space complexities and they are difficult to implement. In order to obtain good timings, it seems to be necessary to use dedicated code generation at different (ranges of) orders n, which can be done using the C++ template mechanism. The current implementation in Mmxlib does not achieve the theoretical time complexity by far, because the recursive function calls suffer from too much overhead.

n	semi, $\mathbb{F}_p$	both, $\mathbb{F}_p$	semi, $\mathbb{C}_{256}$	both, $\mathbb{C}_{256}$
2^8	2.552	2.793	1.481	1.627
2^{10}	2.794	3.423	1.851	2.168
2^{12}	3.486	4.250	2.484	2.987
2^{14}	3.576	4.584	2.757	3.683
2^{16}	3.940	5.135	3.429	4.604
2^{18}	4.293	5.490	3.842	5.418
2^{20}	4.329	5.839		
2^{22}	4.509	6.006		

Table 2
Ratios for the computation of n terms of the exponential of a given series using different types of coefficients. In the first two columns, we use $\mathbb{F}_p$ as our ground field, with $p = 32^{30} + 1$. In the last two columns, we compute with 256 bit complex floats from the MPFR library.

6 Conclusion

We have shown how to improve the complexity of relaxed multiplication in the case when the coefficient ring admits sufficiently many 2^p-th roots of unity. The improvement is based on reusing FFT-transforms of pieces of the multiplicands at different levels of the underlying binary splitting algorithm. The new approach has proved to be efficient in practice (see tables 1 and 2).

For further studies, it would be interesting to study the price of artificially adding 2^p-th roots of unity, like in Schönhage-Strassen's algorithm. In practice, we notice that it is often possible, and better, to "cut the coefficients into pieces" and to replace them by polynomials over the complexified doubles $\mathbb{C}_{52}$ or $\mathbb{F}_p$ with $p = 32^{20} + 1$. However, this approach requires more implementation effort.

Acknowledgement We would like to thank the third referee for his detailed comments on the proof of theorem 4, which also resulted in slightly sharper bounds.

References

Bostan, A., Chyzak, F., Ollivier, F., Salvy, B., Schost, E., Sedoglavic, A., april 2006. Fast computation of power series solutions of systems of differential equation. preprint, submitted, 13 pages.

Brent, R., Kung, H., 1978. Fast algorithms for manipulating formal power series. Journal of the ACM 25, 581–595.

Cantor, D., Kaltofen, E., 1991. On fast multiplication of polynomials over arbitrary algebras. Acta Informatica 28, 693–701.

Cook, S., 1966. On the minimum computation time of functions. Ph.D. thesis, Harvard University.

Cooley, J., Tukey, J., 1965. An algorithm for the machine calculation of complex Fourier series. Math. Computat. 19, 297–301.

Hanrot, G., Quercia, M., Zimmermann, P., 2004. The middle product algorithm I. speeding up the division and square root of power series. AAECC 14 (6), 415–438.

Hanrot, G., Zimmermann, P., Dec. 2002. A long note on Mulders' short product. Research Report 4654, INRIA, available from `http://www.loria.fr/ hanrot/Papers/mulders.ps`.

Karatsuba, A., Ofman, J., 1963. Multiplication of multidigit numbers on automata. Soviet Physics Doklady 7, 595–596.

Knuth, D., 1997. The Art of Computer Programming, 3rd Edition. Vol. 2: Seminumerical Algorithms. Addison-Wesley.

Schönhage, A., Strassen, V., 1971. Schnelle Multiplikation grosser Zahlen. Computing 7 7, 281–292.

Sedoglavic, A., 2001. Méthodes seminumériques en algèbre différentielle ; applications à l'étude des propriétés structurelles de systèmes différentiels algébriques en automatique. Ph.D. thesis, École polytechnique.

Toom, A., 1963. The complexity of a scheme of functional elements realizing the multiplication of integers. Soviet Mathematics 4 (2), 714–716.

van der Hoeven, J., July 1997. Lazy multiplication of formal power series. In: Küchlin, W. W. (Ed.), Proc. ISSAC '97. Maui, Hawaii, pp. 17–20.

van der Hoeven, J., 2002. Relax, but don't be too lazy. JSC 34, 479–542.

van der Hoeven, J., 2003a. New algorithms for relaxed multiplication. Tech. Rep. 2003-44, Université Paris-Sud, Orsay, France.

van der Hoeven, J., August 2003b. Relaxed multiplication using the middle product. In: Bronstein, M. (Ed.), Proc. ISSAC '03. Philadelphia, USA, pp. 143–147.

van der Hoeven, J., 2006a. Newton's method and FFT trading. Tech. Rep. 2006-17, Univ. Paris-Sud, submitted to JSC.

van der Hoeven, J., 2006b. On effective analytic continuation. Tech. Rep. 2006-15, Univ. Paris-Sud.

van der Hoeven et al., J., 2002. Mmxlib: the standard library for Mathemagix. `http://www.mathemagix.org/mml.html`.